From L3 to seL4: What Have We Learnt in 20 Years of L4 Microkernels?

Kevin Elphinstone, Gernot Heiser
NICTA and University of New South Wales
Improving IPC by Kernel Design [SOSP]
1993 IPC Performance

Culprit: Cache footprint [SOSP'95]

Mach

L4

raw copy

i486 @ 50 MHz

©2013 Gernot Heiser, NICTA
## IPC Performance over 20 Years

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th>Processor</th>
<th>MHz</th>
<th>Cycles</th>
<th>µs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>1993</td>
<td>i486</td>
<td>50</td>
<td>250</td>
<td>5.00</td>
</tr>
<tr>
<td>Original</td>
<td>1997</td>
<td>Pentium</td>
<td>160</td>
<td>121</td>
<td>0.75</td>
</tr>
<tr>
<td>L4/MIPS</td>
<td>1997</td>
<td>R4700</td>
<td>100</td>
<td>86</td>
<td>0.86</td>
</tr>
<tr>
<td>L4/Alpha</td>
<td>1997</td>
<td>21064</td>
<td>433</td>
<td>45</td>
<td>0.10</td>
</tr>
<tr>
<td>Hazelnut</td>
<td>2002</td>
<td>Pentium 4</td>
<td>1,400</td>
<td>2,000</td>
<td>1.38</td>
</tr>
<tr>
<td>Pistachio</td>
<td>2005</td>
<td>Itanium</td>
<td>1,500</td>
<td>36</td>
<td>0.02</td>
</tr>
<tr>
<td>OKL4</td>
<td>2007</td>
<td>XScale 255</td>
<td>400</td>
<td>151</td>
<td>0.64</td>
</tr>
<tr>
<td>NOVA</td>
<td>2010</td>
<td>i7 Bloomfield (32-bit)</td>
<td>2,660</td>
<td>288</td>
<td>0.11</td>
</tr>
<tr>
<td>seL4</td>
<td>2013</td>
<td>i7 Haswell (32-bit)</td>
<td>3,400</td>
<td>301</td>
<td>0.09</td>
</tr>
<tr>
<td>seL4</td>
<td>2013</td>
<td>ARM11</td>
<td>532</td>
<td>188</td>
<td>0.35</td>
</tr>
<tr>
<td>seL4</td>
<td>2013</td>
<td>Cortex A9</td>
<td>1,000</td>
<td>316</td>
<td>0.32</td>
</tr>
</tbody>
</table>
Core Microkernel Principle: Minimality

A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e. permitting competing implementations, would prevent the implementation of the system’s required functionality. [SOSP’95]
## Minimality: Source Code Size

<table>
<thead>
<tr>
<th>Name</th>
<th>Architecture</th>
<th>C/C++</th>
<th>asm</th>
<th>total kSLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>i486</td>
<td>0</td>
<td>6.4</td>
<td>6.4</td>
</tr>
<tr>
<td>L4/Alpha</td>
<td>Alpha</td>
<td>0</td>
<td>14.2</td>
<td>14.2</td>
</tr>
<tr>
<td>L4/MIPS</td>
<td>MIPS64</td>
<td>6.0</td>
<td>4.5</td>
<td>10.5</td>
</tr>
<tr>
<td>Hazelnut</td>
<td>x86</td>
<td>10.0</td>
<td>0.8</td>
<td>10.8</td>
</tr>
<tr>
<td>Pistachio</td>
<td>x86</td>
<td>22.4</td>
<td>1.4</td>
<td>23.0</td>
</tr>
<tr>
<td>L4-embedded</td>
<td>ARMv5</td>
<td>7.6</td>
<td>1.4</td>
<td>9.0</td>
</tr>
<tr>
<td>OKL4 3.0</td>
<td>ARMv6</td>
<td>15.0</td>
<td>0.0</td>
<td>15.0</td>
</tr>
<tr>
<td>Fiasco.OC</td>
<td>x86</td>
<td>36.2</td>
<td>1.1</td>
<td>37.6</td>
</tr>
<tr>
<td>seL4</td>
<td>ARMv6</td>
<td>9.7</td>
<td>0.5</td>
<td>10.2</td>
</tr>
</tbody>
</table>
L4 Family Tree

API Inheritance

Code Inheritance

L3 → L4

L4/Alpha

L4/MIPS

“X”

Hazelnut

Pistachio

L4-embed.

seL4

OKL4 Microvisor

OKL4 µKernel

Codezero

Fiasco

Fiasco.OC

NOVA

P4 → PikeOS

UNSW/NICTA

GMD/IBM/Karlsruhe

Dresden

OK Labs

Commercial Clone

©2013 Gernot Heiser, NICTA

SOSP'13
L4 Deployments – in the Billions

SiM Ko 3 “Merkelphone”
seL4: Unprecedented Dependability

Confidentiality

Availability

Integrity

Non-interference [S&P’13]

Abstract Model

Functional correctness [SOSP’09]

Translation correctness [PLDI’13]

C Implementation

Timeliness [RTSS’11, EuroSys’12]

Binary code

• First & only OS kernel with security proofs to binary code
• First & only protected-mode OS kernel with sound timeliness analysis
L4 Design and Implementation

Implement. Tricks [SOSP’93]
- Process kernel
- Virtual TCB array
- Lazy scheduling
- Direct process switch
- Non-preemptible
- Non-portable
- Non-standard calling convention
- Assembler

Design Decisions [SOSP’95]
- Synchronous IPC
- Rich message structure, arbitrary out-of-line messages
- Zero-copy register messages
- User-mode page-fault handlers
- Threads as IPC destinations
- IPC timeouts
- Hierarchical IPC control
- User-mode device drivers
- Process hierarchy
- Recursive address-space construction

Objective: Minimise cache footprint and TLB misses
DESIGN
L4 Synchronous IPC

Rendezvous model

Kernel executes in sender’s context
- copies memory data directly to receiver (single-copy)
- leaves message registers unchanged during context switch (zero copy)
“Long” IPC

- IPC page faults are nested exceptions ⇒ In-kernel concurrency
  - L4 executes with interrupts disabled for performance, no concurrency
- Must invoke untrusted usermode page-fault handlers
  - potential for DOSing other thread
- Timeouts to avoid DOS attacks
  - complexity

Why have long IPC?
- POSIX-style APIs
  - write (fd, buf, nbytes)
- Usually prefer shared buffers
Timeouts

Limit IPC blocking time

Thread\(_1\) Running Blocked
Send (dest, msg)

Thread\(_2\) Blocked Running
Wait (src, msg)

Kernel copy

Rcv (NIL_THRD, delay)

Timed wait

IPC Timeouts ABANDONED in seL4, OKL4

• No theory/heuristics for determining timeouts
• Typically server reply with zero TO, else \(\infty\)
• Can do timed wait with timer syscall
Synchronous IPC Issues

- Sync IPC forces multi-threaded code!
- Also poor choice for multi-core
Asynchronous Notifications

Thread\textsubscript{1} Running → Thread\textsubscript{2} Blocked

Send (Thr\textsubscript{2}, …) → \textit{w} = \texttt{Poll}(…)

\begin{itemize}
  \item Delivers few bits (destructively)
  \item Kernel only buffers single word
  \item Maps well to interrupts, exceptions
\end{itemize}

Thread can wait for synchronous and asynchronous messages concurrently

Server

\begin{itemize}
  \item \texttt{Sync IPC}
  \item \texttt{Async}
\end{itemize}

Client → Sync → Async → Driver

\textbf{Sync IPC complemented with async}
IPC Destination Naming

Original L4 addressed IPC to threads

Client must do load balancing?

RPC reply from wrong thread!

Thread IDs replaced by IPC “endpoints” (ports)

- Inefficient designs
- Poor information hiding
- Covert channels [Shapiro ‘02]

©2013 Gernot Heiser, NICTA
Endpoints

Sync EP queues senders/receivers
Does not buffer messages

Async EP accumulates bits
Other Design Issues

IPC Control: “Clans & Chiefs”

IPC outside clan re-directs to chief

Hierarchies replaced by delegatable cap-based access control

Inflexible, clumsy, inefficient hierarchies!

Process Hierarchy

Hierarchical resource management

Create
IMPLEMENTATION
Virtual TCB Array

- Fast TCB & stack lookup
- Trades cache for TLB footprint and virtual address space

 Trades TLB footprint for cache and kernel memory

- Not worthwhile on modern processors!
- Stacks can dominate kernel memory use!

Move to event kernel

Virtual TCB array ABANDONED
“Lazy” Scheduling

- In IPC-based systems, threads block and unblock frequently
- Many ready queue manipulations

```cpp
thread_t schedule() {
    foreach (prio in priorities) {
        foreach (thread in runQueue[prio]) {
            if (isRunnable(thread))
                return thread;
            else
                schedDequeue(thread);
        }
    }
    return idleThread;
}
```

Idea: leave blocked threads in ready queue, scheduler cleans up

Scheduler execution time is unbounded!

“Benno scheduling”:
- All threads on ready queue are runnable
- All runnable threads in ready queue except the running one
### Implement. Tricks [SOSP’93]

- Process kernel
- Virtual TCB array
- Lazy scheduling
- Direct process switch
- Non-preemptible
- Non-portable
- Non-standard calling convention
- Assembler

### Design Decisions [SOSP’95]

- Synchronous IPC
- Rich message structure, arbitrary out-of-line messages
- Zero-copy register messages
- User-mode page-fault handlers
- Threads as IPC destinations
- IPC timeouts
- Hierarchical IPC control
- User-mode device drivers
- Process hierarchy
- Recursive address-space construction
What are the Principles?

- Minimality is excellent driver of design decisions
  - L4 kernels have become simpler over time
  - Policy-mechanism separation (user-mode page-fault handlers)
  - Device drivers really belong to user level
  - Minimality is key enabler for formal verification!

- IPC speed still matters
  - But not everywhere, premature optimisation is wastive
  - Compilers have got so much better
  - Verification does not compromise performance
  - Verification invariants can help improve speed! [Shi, OOPSLA’13]

- Also found that capabilities are the way to go
  - Shapiro (EROS) was right

- However, a clean abstraction of time still elusive
Conclusions

• Details changed, but principles remained
• Microkernels rock! (If done right!)

Thank you!

We’re hiring:
• Chair in Software Systems
• Postdocs / junior faculty